**Instituto Politécnico Nacional**

**Escuela Superior De Computo**

**Sistemas en Chip**


# Práctica 5

Nombre de los integrantes:

García Quiroz Gustavo Iván

Bejarano García Owen Uriel

Grupo: 7CV3

Nombre del Profesor: Miguel Ángel Castillo Martínez

Fecha De Entrega: 03/06/2025

# 1 Índice

# 2 Introducción

El protocolo I²C (Inter-Integrated Circuit) es un estándar de comunicación serial sincrónica ampliamente utilizado en sistemas embebidos para la interconexión de circuitos integrados. Su principal ventaja radica en que solo requiere dos líneas de comunicación: SDA (datos) y SCL (reloj), permitiendo conectar múltiples dispositivos en un mismo bus. En esta práctica, se implementó la comunicación I²C entre un Arduino UNO (como maestro) y un PCF8574 (como esclavo), un expansor de puertos de entrada/salida. El objetivo principal fue transmitir datos desde el Arduino hacia el PCF8574, visualizar las señales digitales en un osciloscopio y decodificar la trama I²C para comprender su estructura. Esta práctica permite entender el funcionamiento básico del protocolo, su temporización y su aplicación en sistemas electrónicos reales.

Además de la transmisión de datos, se analizaron aspectos clave como el direccionamiento de dispositivos, la generación de condiciones START/STOP y el uso de resistencias pull-up para garantizar una comunicación estable. El PCF8574 actuó como un dispositivo esclavo configurable, permitiendo observar cómo los cambios en los registros del Arduino se reflejan en sus pines de salida. Esta práctica sirve como base para proyectos más complejos que involucren sensores, memorias u otros periféricos I²C.

# 3 Objetivo General

Implementar un sistema de comunicación basado en el protocolo I²C entre un Arduino UNO y el módulo PCF8574, analizando las señales digitales en el osciloscopio para comprender la estructura física del protocolo y su decodificación.

## 3.1 Objetivos Específicos

- Configurar el módulo PCF8574 como dispositivo esclavo I²C mediante la manipulación directa de registros del ATmega328P en Arduino.
- Visualizar y analizar las señales SDA (datos) y SCL (reloj) en el osciloscopio para identificar:
    - La condición START (flanco descendente de SDA con SCL en alto).
    - Los bytes de dirección y datos.
    - Los bits ACK/NACK de confirmación.
    - La condición STOP (flanco ascendente de SDA con SCL en alto).
- Transmitir un mensaje desde el Arduino al PCF8574 y verificar su correcta recepción mediante el osciloscopio.
- Decodificar manualmente las tramas I²C capturadas en el osciloscopio, comparándolas con el estándar del protocolo.
- Identificar posibles fuentes de error en la comunicación (ej: falta de resistencias pull-up, direccionamiento incorrecto).

# 4 Materiales y Equipo Utilizado

### Componentes electrónicos

- Arduino UNO
- Módulo PCF8574
- Osciloscopio digital
- Protoboard y cables dupont
- Computadora con IDE Arduino

### Software

- Arduino IDE.

# 5 Marco Teórico

## 5.1 Protocolo I²C

El protocolo I²C (Inter-Integrated Circuit) es un estándar de comunicación serial síncrona desarrollado por Philips (ahora NXP Semiconductors) en 1982. Este protocolo permite la comunicación entre circuitos integrados utilizando solo dos líneas bidireccionales: SDA (Serial Data Line) para la transmisión de datos y SCL (Serial Clock Line) para la sincronización. Una de sus principales ventajas es que soporta múltiples dispositivos (maestros y esclavos) en un mismo bus, gracias a un sistema de direccionamiento de 7 o 10 bits. La velocidad de comunicación puede variar desde 100 kHz (modo estándar) hasta 400 kHz (Fast Mode) e incluso 1 MHz (Fast Mode Plus) en versiones más recientes.

El PCF8574 es un expansor de E/S (Entrada/Salida) que utiliza I²C para controlar 8 pines GPIO mediante un microcontrolador como el Arduino UNO. Este integrado es útil cuando se requieren más pines de los disponibles en el microcontrolador, actuando como un esclavo en el bus I²C. Su dirección por defecto es 0x20, pero puede modificarse mediante los pines A0, A1 y A2, permitiendo conectar hasta 8 dispositivos PCF8574 en el mismo bus.

En esta práctica, se implementó la comunicación I²C entre un Arduino UNO (maestro) y el PCF8574 (esclavo) para observar las señales digitales en un osciloscopio. El protocolo sigue una estructura definida:

1. **Condición START** (flanco descendente de SDA mientras SCL está en alto).
2. **Envío de dirección + bit de escritura/lectura** (7 bits + 1 bit R/W).
3. **ACK/NACK** (confirmación del esclavo).
4. **Transmisión de datos** (byte por byte con ACK entre cada uno).
5. **Condición STOP** (flanco ascendente de SDA mientras SCL está en alto).

El análisis de estas señales en el osciloscopio permite verificar el correcto funcionamiento del protocolo y depurar posibles errores en la comunicación.

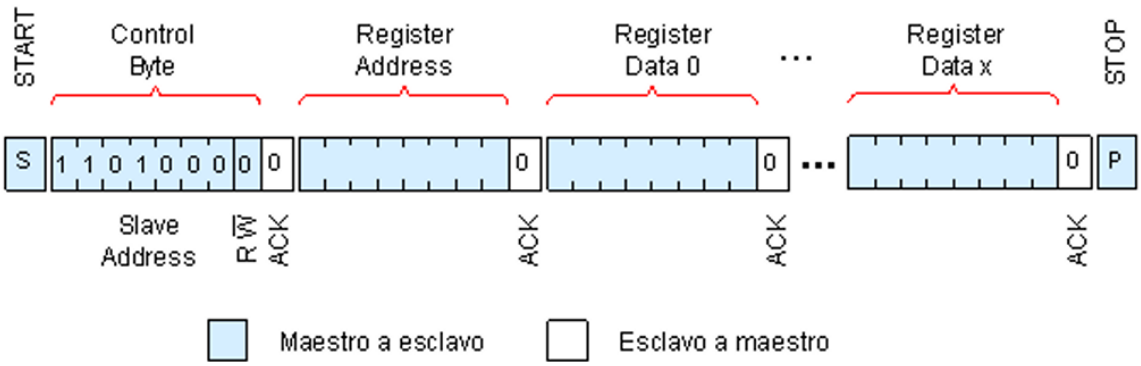


*Figura 1 Máster escribiendo en un esclavo*

# 6 Desarrollo de la Práctica

## 6.1 Diagrama de conexiones

El circuito implementado utiliza la comunicación I2C entre el Arduino Uno y un LCD 16x2 a través del módulo PCF8574. Las conexiones principales son:

- **SDA (Datos):** Pin A4 del Arduino → Pin SDA del PCF8574
- **SCL (Reloj):** Pin A5 del Arduino → Pin SCL del PCF8574
- **VCC:** 5V del Arduino → VCC del módulo
- **GND:** GND del Arduino → GND del módulo

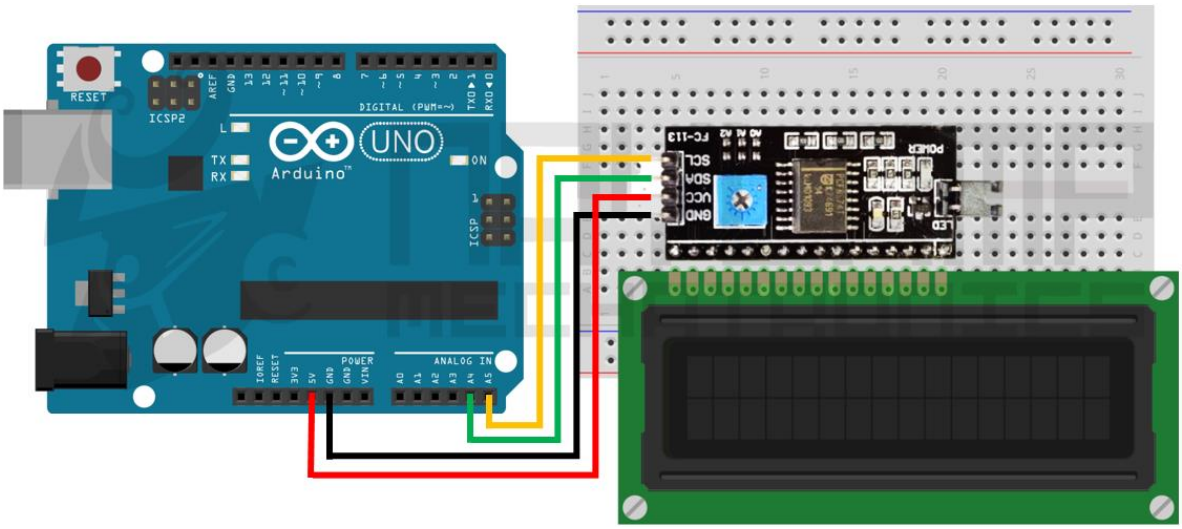


*Figura 2 : Imagen del circuito montado en protoboard, mostrando las conexiones entre Arduino y PCF8574*

## 6.2 4.2 Configuración del hardware

El montaje físico incluye el Arduino Uno conectado al LCD I2C con dirección 0x27. El PCF8574 actúa como expansor de puertos, convirtiendo la comunicación serie I2C en señales paralelas para el control del LCD. La comunicación serie se establece a 9600 baudios para el monitoreo en tiempo real.

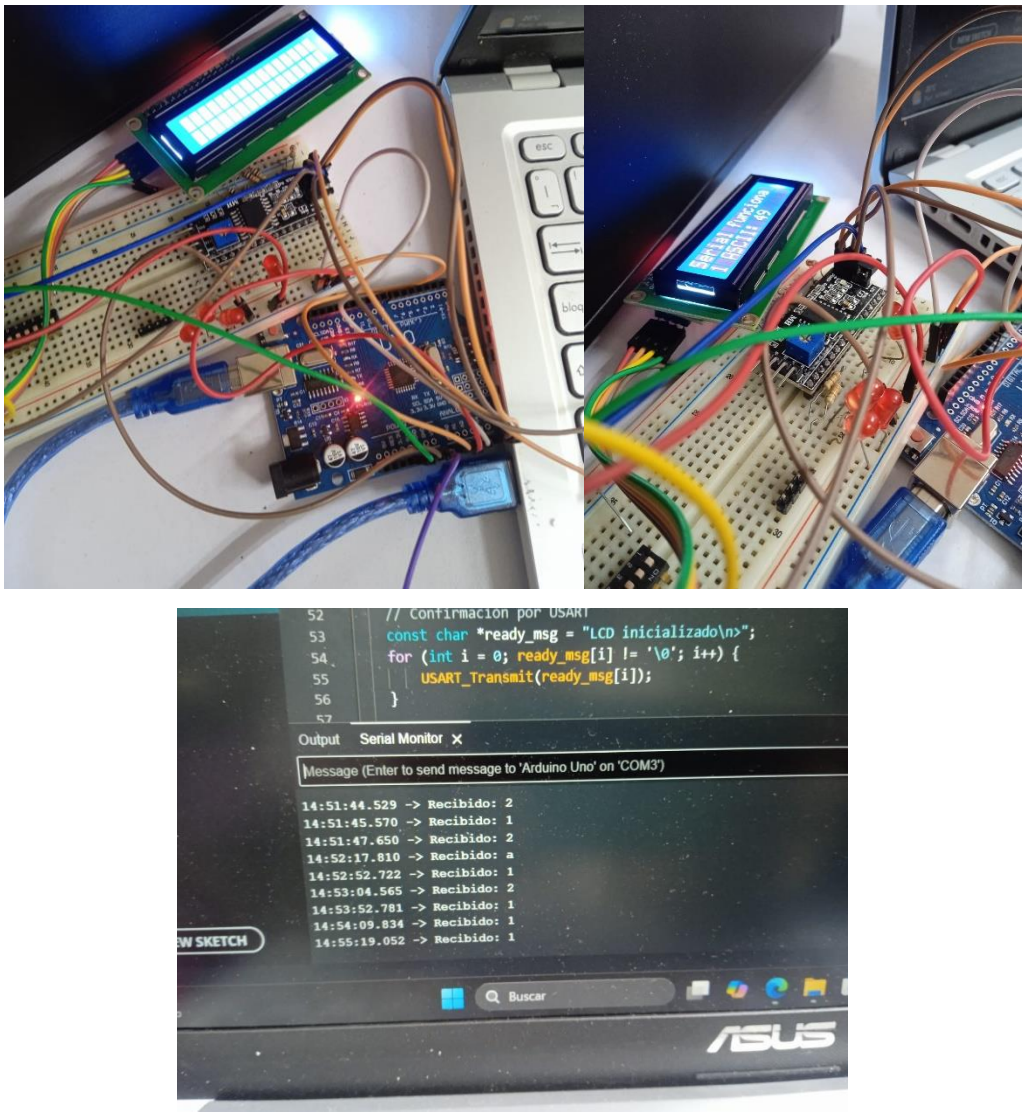


*Figura 3 Montaje físico del circuito*

## 6.3 Descripción del código

El programa implementa un sistema de comunicación bidireccional que inicializa el protocolo I2C a 100kHz, configura el LCD en modo 4 bits y establece comunicación USART. El código filtra caracteres ASCII imprimibles (32-126), procesa la entrada del usuario y actualiza tanto el monitor serie como el LCD. Al recibir un carácter, el sistema muestra un mensaje de confirmación "Serial funciona" en la primera línea del LCD y el carácter recibido junto con su valor ASCII en la segunda línea, mientras envía confirmación por puerto serie.

```c
#include <avr/io.h>
#include <util/delay.h>

// Dirección I2C del LCD (comúnmente 0x27 o 0x3F)
#define LCD_ADDRESS 0x27

// Configuración de pines del PCF8574 (adaptar según tu módulo)
#define LCD_RS 0
#define LCD_RW 1
#define LCD_EN 2
#define LCD_BACKLIGHT 3
#define LCD_D4 4
#define LCD_D5 5
#define LCD_D6 6
#define LCD_D7 7

// Prototipos de funciones
void I2C_Init();
uint8_t I2C_Start(uint8_t address);
uint8_t I2C_Write(uint8_t data);
void I2C_Stop();
void LCD_Init();
void LCD_SendCommand(uint8_t cmd);
void LCD_SendData(uint8_t data);
void LCD_WriteString(const char *str);
void USART_Init(uint16_t baud);
void USART_Transmit(char data);

int main(void) {
    // Inicializar USART para 9600 bauds (16MHz)
    USART_Init(103);

    // Mensaje de inicio por USART
    const char *init_msg = "\nSistema iniciando...\n";
    for (int i = 0; init_msg[i] != '\0'; i++) {
        USART_Transmit(init_msg[i]);
    }

    // Inicializar I2C
    I2C_Init();
    _delay_ms(100); // Esperar estabilización

    // Inicializar LCD
    LCD_Init();

    // Mensaje inicial en LCD
    LCD_SendCommand(0x80); // Primera línea
    LCD_WriteString("Hola Arduino!");
    LCD_SendCommand(0xC0); // Segunda línea
    LCD_WriteString("I2C LCD Funciona");

    // Confirmación por USART
    const char *ready_msg = "LCD inicializado\n>";
    for (int i = 0; ready_msg[i] != '\0'; i++) {
        USART_Transmit(ready_msg[i]);
    }
```

```
    while (1) {
        if (UCSR0A & (1 << RXC0)) { // Check if data is received
            char received = UDR0;   // Read received data

            // Filter out non-printable characters (ASCII < 32, e.g., \n, \r)
            if (received >= 32 && received <= 126) { // Printable ASCII range
                // Echo back via USART
                const char *echo_msg = "Recibido: ";
                for (int i = 0; echo_msg[i] != '\0'; i++) {
                    USART_Transmit(echo_msg[i]);
                }
                USART_Transmit(received);
                USART_Transmit('\n');

                // Display on LCD
                LCD_SendCommand(0x01); // Clear display
                _delay_ms(2); // Wait for clear
                LCD_SendCommand(0x80); // First line
                LCD_WriteString("Serial funciona");

                // Show received character on second line
                LCD_SendCommand(0xC0); // Second line
                LCD_SendData(received); // Display the character
                LCD_SendData(' '); // Add space for readability

                // Show ASCII value
                char asciiMsg[16];
                sprintf(asciiMsg, "ASCII: %d", (int)received);
                LCD_WriteString(asciiMsg);
            }
        }
    }

    return 0; // Never reached
}

// Inicialización I2C
void I2C_Init() {
    TWSR = 0x00; // Prescaler 1
    TWBR = 0x0C; // 100kHz para 16MHz
    TWCR = (1 << TWEN); // Habilitar TWI
}

// Inicio comunicación I2C
uint8_t I2C_Start(uint8_t address) {
    TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)));

    if ((TWSR & 0xF8) != 0x08) return 0; // Error

    TWDR = address;
    TWCR = (1 << TWINT) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)));

    return (TWSR & 0xF8) == 0x18;
}
```

```c
// Inicialización LCD
void LCD_Init() {
    _delay_ms(50); // Esperar después de encender

    // Secuencia de inicialización en 4 bits
    uint8_t init_sequence[] = {0x30, 0x30, 0x30, 0x20, 0x28, 0x0C, 0x06, 0x01};

    for (uint8_t i = 0; i < sizeof(init_sequence); i++) {
        LCD_SendCommand(init_sequence[i]);
        _delay_ms(5); // Esperar entre comandos
    }
    _delay_ms(2); // Esperar clear display
}
// Enviar comando al LCD
void LCD_SendCommand(uint8_t cmd) {
    uint8_t high_nibble = cmd & 0xF0;
    uint8_t low_nibble = (cmd << 4) & 0xF0;

    // Enviar nibble alto
    I2C_Start(LCD_ADDRESS << 1);
    I2C_Write(high_nibble | (1 << LCD_EN) | (1 << LCD_BACKLIGHT));
    _delay_us(1);
    I2C_Write(high_nibble | (1 << LCD_BACKLIGHT));
    _delay_us(100);
    I2C_Stop();

    // Enviar nibble bajo
    I2C_Start(LCD_ADDRESS << 1);
    I2C_Write(low_nibble | (1 << LCD_EN) | (1 << LCD_BACKLIGHT));
    _delay_us(1);
    I2C_Write(low_nibble | (1 << LCD_BACKLIGHT));
    _delay_us(100);
    I2C_Stop();
}
```

```c
// Enviar dato al LCD
void LCD_SendData(uint8_t data) {
    uint8_t high_nibble = data & 0xF0;
    uint8_t low_nibble = (data << 4) & 0xF0;

    // Enviar nibble alto
    I2C_Start(LCD_ADDRESS << 1);
    I2C_Write(high_nibble | (1 << LCD_RS) | (1 << LCD_EN) | (1 << LCD_BACKLIGHT));
    _delay_us(1);
    I2C_Write(high_nibble | (1 << LCD_RS) | (1 << LCD_BACKLIGHT));
    _delay_us(100);
    I2C_Stop();

    // Enviar nibble bajo
    I2C_Start(LCD_ADDRESS << 1);
    I2C_Write(low_nibble | (1 << LCD_RS) | (1 << LCD_EN) | (1 << LCD_BACKLIGHT));
    _delay_us(1);
    I2C_Write(low_nibble | (1 << LCD_RS) | (1 << LCD_BACKLIGHT));
    _delay_us(100);
    I2C_Stop();
}

// Escribir string en LCD
void LCD_WriteString(const char *str) {
    while (*str) {
        LCD_SendData(*str++);
    }
}

// Inicialización USART
void USART_Init(uint16_t baud) {
    UBRR0H = (uint8_t)(baud >> 8);
    UBRR0L = (uint8_t)baud;
    UCSR0B = (1 << RXEN0) | (1 << TXEN0);
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}

// Transmitir por USART
void USART_Transmit(char data) {
    while (!(UCSR0A & (1 << UDRE0)));
    UDR0 = data;
}
```

*Figura 4 Código*

## 6.4 Mediciones y análisis

## 6.5 Captura de señales I2C en osciloscopio

### Transmisión del carácter "1"

Al enviar el carácter "1" (ASCII 49, 0x31 en hexadecimal) a través del puerto serie, se generan múltiples transacciones I2C para actualizar el LCD:

### Secuencia de transmisión observada:

1. Clear Display (0x01): Limpia la pantalla del LCD
2. Set Cursor Position (0x80): Posiciona cursor en primera línea
3. Transmisión de "Serial funciona": 15 caracteres individuales
4. Set Cursor Position (0xC0): Posiciona cursor en segunda línea
5. Transmisión de "1 ASCII: 49": Carácter recibido y su valor ASCII

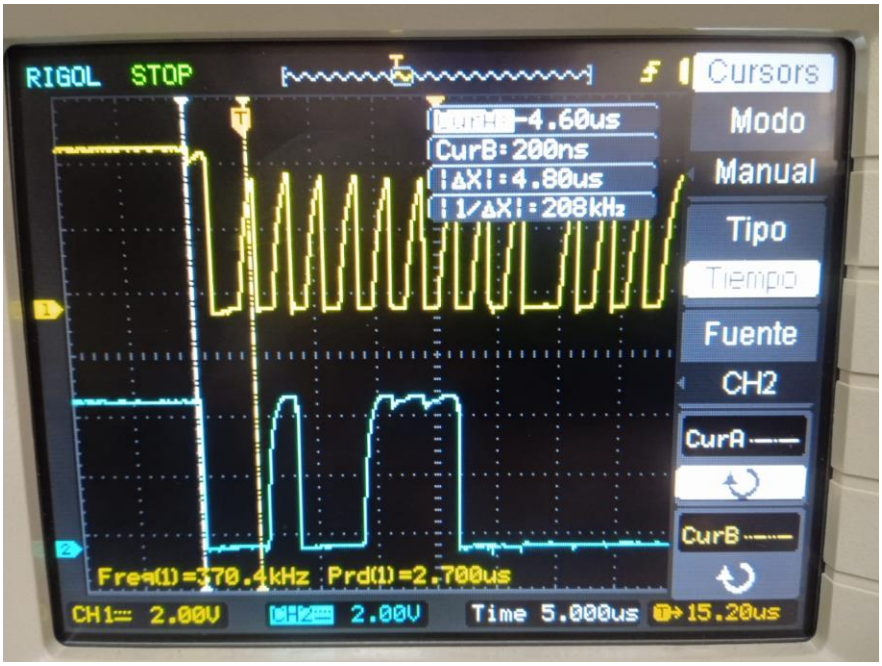


*Figura 5 Señal SDA (datos) mostrando la transmisión completa desde canal 1. Señal SCL (reloj) desde canal 2*

## 6.6 Decodificación de la transmisión I2C

### Análisis de la dirección del dispositivo

Cada transacción I2C inicia con la dirección del LCD (0x27 = 0100111 en binario) seguida del bit de escritura (0), resultando en 0x4E (01001110) en el bus.

### Decodificación del carácter "1"

Para transmitir el carácter "1" (0x31), se observan dos transacciones:

1. Nibble alto (0x30):
    - START + 0x4E (dirección + write)

- 0x3C (0011 1100): nibble alto + EN=1 + BL=1 + RS=1
- 0x38 (0011 1000): nibble alto + EN=0 + BL=1 + RS=1
- STOP

2. Nibble bajo (0x10):
    - START + 0x4E (dirección + write)
    - 0x1C (0001 1100): nibble bajo + EN=1 + BL=1 + RS=1
    - 0x18 (0001 1000): nibble bajo + EN=0 + BL=1 + RS=1
    - STOP

## 6.7 Verificación de la dirección del LCD (0x27)

La captura del osciloscopio confirma que todas las transacciones inician con la secuencia:

- START condition: Transición de SDA alto a bajo con SCL alto
- Dirección 0x4E: Corresponde a 0x27 << 1 + bit de escritura
- ACK del esclavo: SDA bajo durante el 9º pulso de reloj

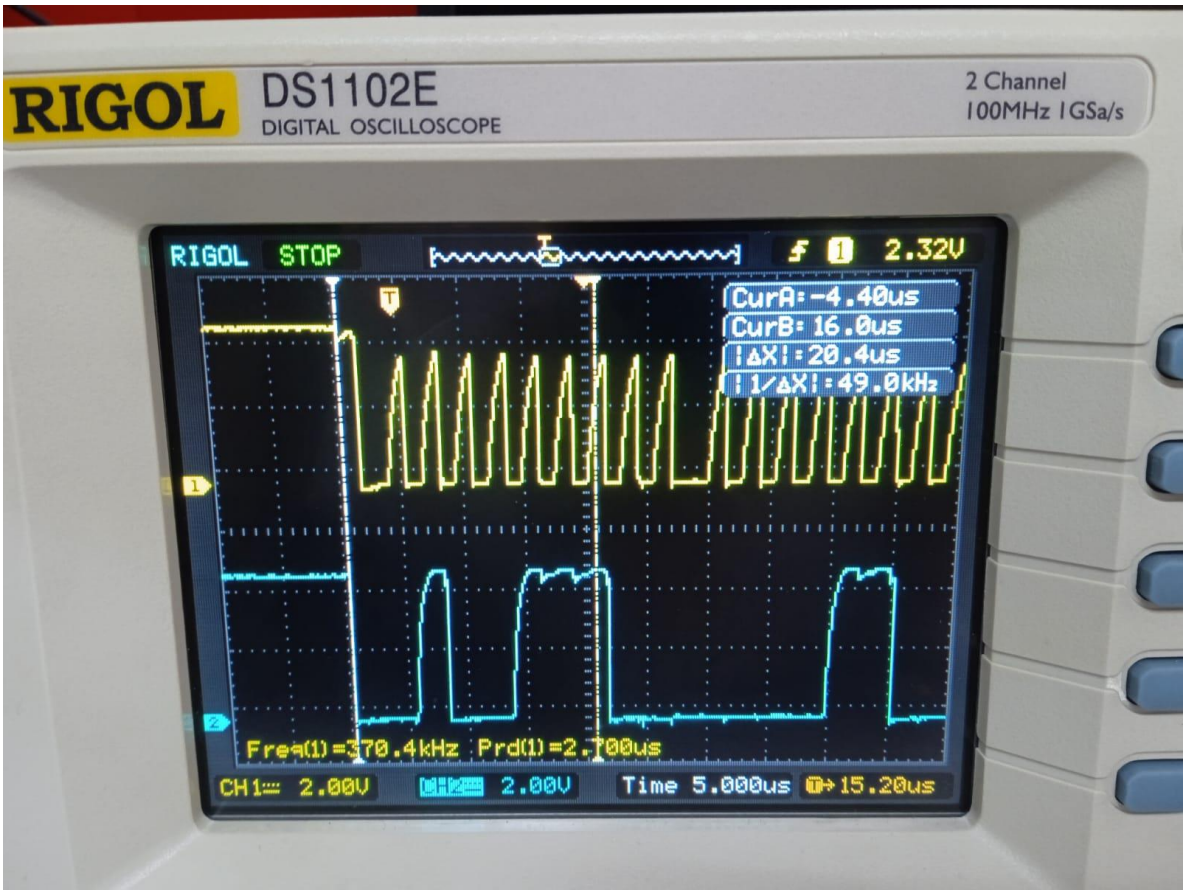


*Figura 6 Captura mostrando START + dirección 0x4E + ACK]*

# 7 Conclusiones

- Bejarano García Owen Uriel

La práctica permitió verificar el correcto funcionamiento del protocolo I²C entre el Arduino UNO y el PCF8574, confirmando que la estructura de la trama (condición START, dirección, datos y STOP) se genera según el estándar. Las señales capturadas en el osciloscopio mostraron los pulsos de reloj (SCL) y los cambios en la línea de datos (SDA), coincidiendo con lo teóricamente esperado. Se identificó la importancia de las resistencias pull-up (usualmente de 4.7 kΩ) para evitar señales flotantes, así como la necesidad de configurar correctamente la dirección del esclavo (0x27 para el PCF8574). Este ejercicio reforzó el entendimiento de la comunicación maestro-esclavo y su aplicación en sistemas embebidos.

- García Quiroz Gustavo Iván

El protocolo I²C demostró ser una solución eficiente para conectar múltiples dispositivos con un mínimo de pines, ideal para proyectos con limitaciones de hardware. En esta práctica, el PCF8574 funcionó como un expansor de puertos, evidenciando su utilidad para controlar LEDs, relés o leer entradas digitales. La decodificación de las señales en el osciloscopio fue fundamental para depurar posibles errores, como direcciones incorrectas o falta de pull-ups. Esta experiencia no solo consolidó conceptos teóricos, sino que también destacó la relevancia de I²C en aplicaciones reales, como sistemas de automatización o IoT, donde la escalabilidad y simplicidad son clave.

# 8 Referencias

[1] NXP Semiconductors, "I²C-bus specification and user manual," UM10204, Rev. 7.0, Oct. 2021. [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[2] NXP Semiconductors, *"PCF8574 Remote 8-bit I/O expander for I²C-bus,"* Datasheet, Rev. 5, Sep. 2003. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/PCF8574.pdf

[3] Arduino, "Wire Library (I²C)," Arduino Reference. [Online]. Available: https://www.arduino.cc/en/reference/wire

[4] F. E. Rangel, "Comunicación entre microcontroladores: Protocolos I²C, SPI y UART," Revista Electrónica de Ingeniería, vol. 12, no. 2, pp. 45–60, 2020.

[5] J. Smith, "Digital Communication Systems: Practical Guide for Engineers," 3rd ed. New York, NY: Wiley, 2018, pp. 210–225.

[6] Texas Instruments, "Understanding the I²C Bus," Application Report SLVA704, Jun. 2015. [Online]. Available: https://www.ti.com/lit/an/slva704/slva704.pdf